CHAPTER OBJECTIVE

- Use the for loop to build basic counting structures
- Modify the for loop for different kinds of counting
- Use a while loop for more flexible looping
- Identify the keys to successful loops
- Create basic arrays
- Write programs that use arrays and loops
- Use session variables to store data
- Store information in hidden fields
Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.

In PHP, we have the following looping statements:

- **while** - loops through a block of code as long as the specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array
FOR LOOPS

The for loop is used when you know in advance how many times the script should run.

Parameters:

- **init counter**: Initialize the loop counter value. $x$ is the Sentry Variable (Page 104)
- **test counter**: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- **increment counter**: Increases the loop counter value

Notation

The notation $i++$ is just like saying add one to $i$ or $i = i + 1$. The ++ symbol is called an increment operator because it provides an easy way to increment (add 1) to a variable.

```php
<?php
for ($x = 0; $x <= 10; $x++) {
    echo "The number is: $x <br";
}
?>
```

Note the braces around the code to be executed inside the for loop. You can have as many lines of code as you want.
FOR LOOPS

for (initial expression; condition; closing expression) {
    // Do something.
}

```php
<?php
for ($i = 1; $i <= 10; $i++) {
echo $i;
} // end for loop
?>
```

The for loop uses a more complicated syntax.
The initial expression is only executed once.
The closing expression is executed once for each loop iteration.
The for loop is normally used when you’ll do something a known number of times.

$i = 1 //initial $i to one
$i <= 10 // while $i is less than or equal to ten
$i++ //increment $i with each loop

Execute echo command to print value of $i
For loop will end when $i = 11 since the condition $i <= 10 (11<= 10) is now FALSE
MODIFYING THE FOR LOOP

<?php
print "<p> \n";
for ($i = 5; $i <= 50; $i+= 5){
    print "$i <br />\n";
} // end for loop
print "</p> \n";
?>

Changes made to various parameters in the for statement. The code starts counting at 0. The initial value set for $i to 5. For loop stops when $i reached 50 (after 10 iterations). Each time through the loop, $i is incremented by five.

The += syntax in the following code increments a variable:$i += 5; The above is the same thing as this: $i = $i + 5;

What if we want to go backwards?

<?php
print "<p> \n";
for ($i = 10; $i > 0; $i--){
    print "$i <br />\n";
} // end for loop
print "</p> \n";
?>
The while loop executes a block of code as long as the specified condition is TRUE, once the condition is FALSE, the loop ends. (Page 110)

The example below first sets a variable $x to 1 ($x = 1). Then, the while loop will continue to run as long as $x is less than, or equal to 5 ($x <= 5). $x will increase by 1 each time the loop runs ($x++):

```php
<?php
$x = 1;
while($x <= 5) {
    echo "The number is: $x <br>";
    $x++;
}
?>
```
WHILE LOOPS

while (condition) {
    // Do something.
}

When you write a while loop, you are responsible for these three things:

• Creating a sentry variable - $i$ variable
• Building a condition statement
• Ensuring the loop can exit

A while loop runs so long as a condition is true.

Normally used when it’s not known in advance how many times the loop is needed (e.g., when retrieving database query results).
BUILDING A WELL-BEHAVED LOOP

Create/Initializing a Sentry Variable

- If your loop is based on a variable’s value (there are alternatives), make sure you do these three things:
  - Identify the variable.
  - Ensure the variable has appropriate scope.
  - Make sure the variable has a reasonable starting value.

Condition of the Loop/Loop Exit

- Your condition usually compares a variable and a value. Make sure you have a condition that can be met and be broken. The hard part is ensuring that the program gets out of the loop at the correct time. This condition is much like the condition in the for loop.
- There must be some trigger that changes the sentry variable so the loop can exit. This code must exist inside the code body. Be sure it is possible for the sentry variable to achieve the value necessary to exit the loop by making the condition false.
The flexibility of the while construct gives it power, but with that power comes potential for problems.

While loops are easy to build, but a loop that works improperly can cause a lot of trouble. It’s possible that the code in the loop will never execute at all. Even worse, you might have some sort of logical error that causes the loop to continue indefinitely. (Page 112)

```php
<?php
$i = 1;

while ($i <= 10){
    print "$i <br />\n";
    $i++;
} // end while
?>
</div>
```

i is never incremented. Which means $i will always equal one and the condition $i <= 10 will always be true. This is an endless loop, it will print 1 forever.
An array stores multiple values in one single variable:

- Arrays hold multiple values in one variable
- Series of key-value pairs
- Indexed arrays use numbers for keys
- Associative arrays use strings
- Indexed arrays begin at 0 by default
If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```php
$cars1 = "Volvo";
$cars2 = "BMW";
$cars3 = "Toyota";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is to create an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.
In PHP, there are three types of arrays:

- Indexed/Basic arrays - Arrays with a numeric index
- Associative arrays - Arrays with named keys
- Multidimensional arrays - Arrays containing one or more arrays

For now we will only cover Index/Basic Arrays
The $camelPop variable is a variable meant to hold the five countries with the largest camel populations in the world.

Since $camelPop is going to hold the names of five different countries, it makes sense that this is an array (computer geek lingo for list) rather than an ordinary variable. (Page 116)
Arrays go naturally with for loops. Very often when you have an array variable, you step through all of its values and do something to each one.

I know the array indices ($i$) will vary between 1 and 5, the loop value of $i$ will go from 1 to 5. Inside the loop, simply print the index ($i$) and the corresponding country ($\text{camelPop}[$i$]$).

The first time through the loop, $i$ is 1, so $\text{camelPop}[$i$]$ is $\text{camelPop}[1]$, which is Somalia. Each time through the loop, the value of $i$ is incremented, so eventually every array element is displayed.

```php
for ($i = 1; $i <= 5; $i++)
{
    print "$i: $\text{camelPop}[$i$]<br />\n";
} // end for loop
```
ARRAY TRICKS

Preload an Array

- Often you start out knowing exactly which values you want placed in an array. PHP provides a shortcut for loading an array with a set of values.

```php
//use array function to load up array
$binary = array("000", "001", "010", "011");
```

Detecting the Size of an Array

- PHP provides a function called `count()`, which can determine how many elements an array has.

```php
for ($i = 0; $i < count($binary); $i++) {
    print "$i: $binary[$i]<br />

} // end for loop
```
ACCESSING ARRAYS

// Use array notation to access individual elements:
$band = $artists[0];
echo $states['MD'];

// You can’t do this:
echo "My list of states: $states";    // BAD!

// Or this:
echo "IL is $states['IL'].";    // BAD!

Parse error: syntax error, unexpected T_ENCAPSED_AND_WHITESPACE,
expecting T_STRING or T_VARIABLE or T_NUM_STRING in
/Users/larryullman/Sites/phpmysql4/test.php on line 13
- Arrays can be created by adding an element at a time. You can specify the key or not.
- Arrays can also be created using the array() function. Again, you can specify the keys or not.
LOOPING THROUGH ARRAYS

- foreach ($array as $value) {
  // Do something with $value.
  }

  // Or:

  foreach ($array as $key => $value) {
    echo "The value at $key is $value.";
    }

SORTING ARRAYS

- `sort()`, by value, dropping the keys
- `asort()`, by value, maintaining the keys
- `ksort()`, by key, maintaining the values
- `rsort()`, `sort()` in reverse order
- `arsort()`, `asort()` in reverse order
- `krsort()`, `ksort()` in reverse order
$array = explode (separator, $string);
$string = implode (glue, $array);

$s1 = 'Mon-Tue-Wed-Thu-Fri';
$days_array = explode ('-', $s1);
// $days_array now a five-element array, with Mon indexed at 0.

$s2 = implode (',', $days_array);
// $s2 now a comma-separated list of days: Mon, Tue, Wed, Thu, Fri.
Sometimes your programs will need to pass an array of data from an HTML form.

Example: The form is designed so all elements form an array, which can be read by the program when the user clicks the Submit button.
The code begins by detecting whether this is the first time the user has come to the form. This is done by using the `filter_has_var()` function to check for the existence of the `people` variable (which will be the only variable produced by the page). If this variable exists, control is passed to the `showPeople()` function, which will display the contents of the `people` array. If the variable does not exist, this must be the first pass through the program, so the `printForm()` method creates the user input form.
The `printForm()` function basically creates an HTML form. It's nothing but a huge hereDoc.

Note that all the text boxes have a similar name attribute, but they use the array syntax (people[4] for example).
SHOWPEOPLE()

- The `filter_input_array()` function reads all the elements of a form into an array. You can then extract any of the variables from this array.
- The `filter_input_array()` function pulls all of the form elements into a single variable. This can be used with any type of data, but it’s especially handy with arrays.
- You can then extract any of these elements into their own variables. In this case, it isolates the people array and passes it to its own variable:

```php
$people = $formData['people'];
```

`print_r()` function is used to print out complex variables such as arrays.
You can store information a couple of ways, including files, XML, and databases.

The easiest approach to achieving data permanence is to hide the data in the user’s page using a special trick called session variables.
When the program begins, it grabs the value of $txtBoxCounter (if it exists) and adds one to it.

When the program prints the text box, it automatically places the $txtBoxCounter value in the text box.

```html
//increment the counters
$txtBoxCounter++;
$hdnCounter++;

print <<<HERE
  <fieldset>
    <input type = "text"
          name = "txtBoxCounter"
          value = "$txtBoxCounter" />

    <input type = "hidden"
           name = "hdnCounter"
           value = "$hdnCounter" />
  </fieldset>

<h3>The hidden value is $hdnCounter</h3>
<input type = "submit"
       value = "click to increment counters" />
Since the form’s action attribute is set to null (""), the program automatically calls itself when the user clicks the Submit button. This time, $txtBoxCounter has a value (1). When the program runs again, it increments $txtBoxCounter and stores the new value (now 2) in the text box.
When you create a session variable, PHP creates a file on the server containing whatever information is in the session variable, and it creates a random identifier of the session, which is stored in a cookie on the client’s machine.

A cookie is a small text file stored by the browser.

Cookies have gotten a bad reputation, which is largely undeserved. It’s true that websites often use cookies to store data on a user’s machine without explicit permission, but that’s not always a bad thing.
Cookie information must be sent to the client before any text is written to the page. The session_start() code should be the very first line of your PHP program. If there is any HTML or PHP in front of this code, the session will not work correctly.

Session data is stored in a special variable called $_SESSION. The isset() function can be used to tell you if the session already has a value, returning true if the session has a value, or false if it does not.

Use a special syntax to create a regular variable from the session. This line does the work:

```php
$counter = $_SESSION["counter"];`
Starting up the Game

The `startGame()` function gives the user an initial cash balance of $100. The `$cash` variable will need to be tracked over several runs of the program, so it is stored in a session variable. ...

```php
function startGame()
{
    // if it's the first time here, set up initial cash,
    // and do firstPass
    $cash = 100;
    $_SESSION["cash"] = 100;
    firstPass();
} // end reset
```
KEEPING PERSISTENT DATA

- HTTP is a stateless protocol - the webserver does not keep track of a script's state after it has completed.
- All variables in your script are destroyed when the script ends. So, if you need to retain the value of a variable, you will need to save it somehow. We say the data needs to persist, or have persistence.
- You'll often need a mechanism for determining the current state of your web app.
- There are several mechanisms: form fields, cookies, session variables, files, databases.
Solving a problem differs from following someone else's solution to a problem, just as writing a story differs from reading a story. You need analytical skills, imagination, and perseverance.

It helps to look at a lot of examples. Unfortunately, a lot of the examples in tutorials and reference manuals are designed only to illustrate the particular statement or language feature being discussed. Typically, your homework assignments require you to do something more complex and challenging.

Top-down structured design and programming is a powerful, tried and true method for solving even the most complex problems. Think of what needs to be done. Use the divide-and-conquer approach to break the problem down into several smaller problems. Also, consider what tools you have to work with, so you don't end up reinventing the wheel. For example, if you need to sort some items and a sort function already exists, it would be much better to use the existing sort function than to write your own.
We have covered looping, for loops, while loops, arrays, array() construct, stepping through an array with a loop, storing data in form fields for persistence, and have put all these together to create an interesting game.

We have looked at how to go about solving a problem.